INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

State Merging Inference of Finite State Classifiers

François Coste, allocataire MENRT

No 3695

Mai 1999

THÈME 3

ISSN 0249-6399

State Merging Inference of Finite State Classifiers

François Coste, allocataire MENRT

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Aïda

Rapport de recherche n ° 3695 — Mai 1999 — 24 pages

Abstract: In this report, we introduce the C-regular inference problem. It consists in inducing a set of C regular languages from samples of each language. An implicit representation of the search space for state merging inference methods is presented. This representation allows to study more precisely the languages interactions. A characterization of deterministic and unambiguous classifiers in the search space is given and used to design a new state merging algorithm considering not only possible mergings but also impossible ones.

Key-words: Regular inference, Finite State Machine, Classification, State merging algorithms.

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Inférence d'automates classifieurs par fusions d'état

Résumé : Nous introduisons dans ce rapport le problème de l'inférence C-régulière qui consiste à apprendre un ensemble de C langages réguliers à partir d'exemples de chacun des langages. Nous proposons une représentation implicite de l'espace de recherche pour les algorithmes par fusion d'état permettant l'étude directe des interactions entre langages et, notamment, une caractérisation par contraintes des automates classifieurs déterministes et univoques. Cette caractérisation est utilisée pour proposer un nouveau schéma d'algorithme par fusion considérant non seulement les fusions possibles mais également les fusions impossibles.

Mots-clé : Inférence régulière, machines à états finis, classification, algorithmes par fusions d'états.

Note : Ce rapport a été soumis à publication en décembre 1998 au Machine Learning Journal, numéro spécial Inférence Grammaticale (V. Honavar and C. de la Higuera eds).

1. Introduction

The regular inference problem is that of inducing a regular language or its acceptor from a set of examples of the language. Many empirical algorithms using only a positive sample have been devised [12]. For the identification of a target language, a negative sample is needed [11]. Most of the inference algorithms from positive and negative samples proceed by generalizing the positive sample under the control of the negative sample [22, 19, 14, 16, 8, 1, 20, 6, 15].

For several applications in language processing, genetics, information retrieval, syntactic pattern recognition, system security and medical monitoring, generalizing the positive sample solely is not adequate. Alquezar and Sanfeliu propose in [4] to consider symmetrically the positive and the negative samples. In this case, the problem is not the inference of an acceptor of the positive language but rather the inference of a sequence classifier. The classification of a sequence may then be positive, negative or unknown. This can be extended from two languages to C languages: we call *C-regular inference* the problem of inferring simultaneously C regular languages from samples of each language. The originality of C-regular inference is that interactions between languages may be taken into account during the inference process. The fundamental interaction studied in this article is the intersection of languages. More precisely, we address the problem of inferring *unambiguous* set of languages, in the sense that each sequence may be classified in only one language.

The chosen finite state classifier representation of a C-tuple of languages, called *C-classes finite state unbiased automata* (C-FSA) [3], is introduced in section 2. Section 3 is devoted to the study of the search space under the assumption that the sample is structurally complete with respect to the target machine. In this case, the search space is a lattice based on a particular C-FSA MCA such that the elements of the lattice are obtained by merging states of MCA. Since merging two states is the canonical operation for exploring the lattice, we introduce a new implicit search space based on state pair mergings called *state merging space*. The study of this search space allows to characterize deterministic and unambiguous classifiers in the lattice. This characterization may be used by inference algorithms exploring intensively the search space. We present, in section 4, a new state merging algorithm considering not only the possible mergings but also the impossible ones for the inference of deterministic unambiguous classifiers.

2. Finite state classifier

Before introducing C-FSA as a type of finite state Moore machine in section 2.2, we recall some definitions and properties of sequential machines.

2.1. Sequential machines

Informally, a sequential machine is an abstract device that reads a sequence of input symbols and writes an associated sequence of output symbols under the control of a set of states. In this section, we introduce some notations and recall some definitions related to sequential machines presented in [5].

Definition 1. A *sequential machine* (Mealy machine) is characterized by the following:

- A set Q of states.
- A finite alphabet of input symbols Σ_i.
- A finite alphabet of output symbols Σ_o.
- A mapping δ of $Q \times \Sigma_i$ into Q called the *next-state function.*
- A mapping ω of $Q \times \Sigma_i$ into Σ_o called the *output function.*

We denote a particular machine by the 5-tuple $\langle Q, \Sigma_i, \Sigma_o, \delta, \omega \rangle$. If the set of states Q is finite and an initial state q_s is chosen, the sequential machine is called a *finite state machine.*

The type of the output function may be simplified or optimized according to the application properties. For instance, when the output function depends only on the next state, one can define the output mapping ω as restricted to a mapping of Q into Σ_o. This machine is then called a *Moore machine.* By considering the extension of the next-state function and the output function to functions over sequences of symbols, other specialized machines can be defined.

In what follows, q, σ, a and ϵ denote respectively, a state in Q, a sequence in Σ_i^*, a symbol in Σ_i and the empty word. The *state sequence function* $\delta^* : Q \times \Sigma_i^* \to Q^*$ is then recursively defined by

$$\delta^*(q, \epsilon) = q, \ \delta^*(q, a\sigma) = q.\delta^*(\delta(q, a), \sigma).$$

Similarly, the *output sequence function* $\omega^* : Q \times \Sigma_i^* \to \Sigma_o^*$ can be defined as follows.

$$\begin{array}{c} \text{Mealy machine: } \omega^*(q, a\sigma) = \omega(q, a).\omega^*(\delta(q, a), \sigma) \\ \text{Moore machine: } \omega^*(q, \epsilon) = \omega(q), \ \omega^*(q, a\sigma) = \omega(q).\omega^*(\delta(q, a), \sigma). \end{array}$$

From these definitions, one can derive two important functions. The *terminal state function* $\hat{\delta} : Q \times \Sigma_i^* \to Q$ indicating the state of the system after the input sequence has been applied and the *last output function* $\hat{\omega} : Q \times \Sigma_i^* \to \Sigma_o$ describes the last output symbol produced by the machine for the input sequence, are introduced:

$$\hat{\delta}(q, \epsilon) = q, \ \hat{\delta}(q, \sigma a) = \delta(\hat{\delta}(q, \sigma), a)$$

$$\text{Mealy machine: } \hat{\omega}(q, \sigma a) = \omega(\hat{\delta}(q, \sigma), a)$$
$$\text{Moore machine: } \hat{\omega}(q, \sigma) = \omega(\hat{\delta}(q, \sigma)).$$

According to the output function type, several sequential machine sub-families can be characterized. For example, finite state automata are Moore machine with a two output symbol alphabet $\{0, 1\}$, such that each state is assigned output 1 if it is final and output 0 else. The acceptance of a sequence is given by the last output function. In the next section, we introduce a sub-family of finite state machines, using also the last output function but allowing more than two symbols.

Two finite state machines may be compared in terms of their input-output characteristics. When considering finite state machines, the default state for a next-state or an output function is the initial state q_s. Thus, the equivalence of two finite state machines is defined by:

Definition 2. Two finite state machines M and M' are said to be *equivalent* if and only if for each input sequence in Σ_i^* both machines return the same outputs.

One fundamental result related to finite state machines is the existence of a minimization algorithm mapping each finite state machine into an equivalent finite state machine of the same sub-family, with a minimal number of states. This machine is proven to be unique up to a renaming of the states [5].

2.2. C-classes unbiased finite state automata

In [4] the unbiased finite state automata have been introduced to deal in a symmetrical manner with positive and negative data. The author suggests in [3] that this concept could be generalized naturally to cope with C-classes recognition problems and introduces some basic definitions of an extended type of automata called C-classes unbiased finite state automata (C-FSA). For a better understanding of the links with other machines like transducers, this section introduces C-FSA as a type of Moore machines, extended to non-deterministic next-state and output functions, such that the *classification* of a sequence is given by the last output symbol function.

Definition 3. A *C-classes unbiased finite state automata (C-FSA)* is a 6-tuple $(Q, q_s, \Sigma, \Gamma, \delta, \omega)$ where:

- Q is a finite set of states;
- $q_s \in Q$ is the initial state;
- Σ is a finite alphabet of input symbols;
- Γ is a finite alphabet of C output symbols;

- δ is the next-state function mapping $Q \times \Sigma$ to 2^Q;
- ω is the output function mapping $Q \times \Sigma$ to 2^Γ.

The last output function $\hat{\omega}$ which maps $Q \times \Sigma^*$ to 2^Γ allows to define the C-tuple of regular languages $\mathcal{L}(M)$ accepted by a C-FSA M:

$$\mathcal{L}(M) = \langle L_c(M) \rangle_{c \in \Gamma} \ \ where \ \ \forall c \in \Gamma, \ L_c(M) = \{\sigma \in \Sigma^* | c \in \hat{\omega}(q_s, \sigma)\}.$$

The set of sequences over the input alphabet Σ^* is called the *domain* of M. The *scope* of M, denoted by $L_S(M)$, is the language:

$$L_S(M) = \bigcup_{c \in \Gamma} L_c(M)$$

which contains all the sequences accepted by M. The sequences of the complementary language $L_I(M) = \Sigma^* - L_S(M)$ are said to be ignored by M. $L_I(M)$ may be also defined as the set:

$$L_I(M) = \{\sigma \in \Sigma^* \mid \hat{\delta}(q_s, \sigma) = \emptyset \ \vee \ \hat{\omega}(q_s, \sigma) = \emptyset\}$$

using the convention that any undefined function for an input symbol returns the empty set.

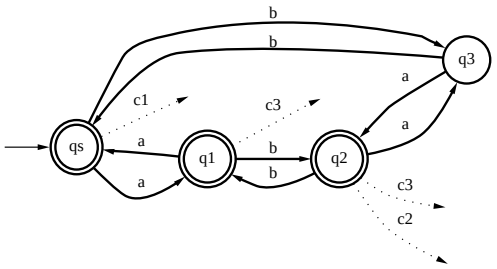

Figure 1. 3-DFA classifying sequences in classes $c1$, $c2$ and $c3$ respectively defined by: the sequence has an even number of a and a even number of b ($c1$), an odd number of a and an odd number of b ($c2$) an odd number of a ($c3$).

Definition 4. A C-FSA is said to be *ambiguous* if a sequence of its domain can be classified in more than one class i.e. $\exists i, j \in \Gamma, i \neq j, \ L_i(M) \cap L_j(M) \neq \emptyset$.

Definition 5. If for any state q in Q and any input symbol a in Σ, the next-state function returns at most one (respectively exactly one) state, the C-FSA is said to be *deterministic* (respectively *complete*). C-DFA and C-NFA denote respectively the deterministic and non deterministic C-FSA.

THEOREM 1 *For each C-FSA M, there exists a C-DFA M_D accepting the same languages, i.e. such that $\mathcal{L}(M) = \mathcal{L}(M_D)$.*

Proof: The proof is given in [3]. It consists in adapting to C-DFA the NFA determinization algorithm [2]. ■

Definition 6. Given a C-tuple of languages $\mathcal{L}$, the *canonical C-FSA* of $\mathcal{L}$, denoted by $M(\mathcal{L})$, is the minimal C-DFA accepting exactly $\mathcal{L}$.

2.3. Derived C-FSA

Definition 7. For any set S, a partition π is a set of pairwise disjoint nonempty subsets of S whose union is S. Let s denote an element of S and let $B_\pi(s)$ denote the unique block of π containing s.

Definition 8. Given a C-FSA $\mathrm{M} = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ and a partition π of the set of states Q, the *C-FSA derived from M with respect to π* is $\mathrm{M}/\pi = (Q', q_{s\pi}, \Sigma, \Gamma, \delta', \omega')$ defined as follows:

- $Q' = \{B_\pi(q) |\ q \in Q\}$
- $\forall B \in Q',\ \omega'(B) = \bigcup_{q \in B} \omega(q)$
- $\forall B \in Q', \forall a \in \Sigma,\ \delta'(B, a) = \{B' \in Q' |\ \exists q \in B, \exists q' \in B', q' \in \delta(q, a)\}$.

the states of Q belonging to the same block B of the partition π are said to be *merged* together.

The operation of merging two states defines a partial order relation denoted by $\preceq$. The transitive closure for this relation is denoted by $\ll$. $M \ll M'$ means that M' is a C-FSA derived from M. The partially ordered set of C-FSA derived from M is a lattice that we shall denote by $Lat(M)$.

The language inclusion property for automata derivation [10] applied to each language of a C-FSA leads to the following proposition.

PROPOSITION 1 *Let M' be a C-FSA accepting the C-tuple of language $\langle L'_c \rangle_{c \in \Gamma}$ such that M' is derived from a C-FSA M accepting $\langle L_c \rangle_{c \in \Gamma}$, then $\forall c \in \Gamma, L_c \subseteq L'_c$.*

3. Search space

We assume that a training sample $\mathcal{S} = \langle S_c \rangle_{c \in \Gamma}$ is given such that each S_c is a sample from each target language $L_c(M)$, i.e. a subset of $L_c(M)$. We denote $\{\sigma_{c,1}, \ldots, \sigma_{c,|S_c|}\}$ the sequences in S_c and $\{a_{c,i,1}, \ldots, a_{c,i,|\sigma_{c,i}|}\}$ the sequence of symbols in $\sigma_{c,i}$.

To define the search space, a classical assumption in regular inference is to suppose that the sample is structurally complete with respect to the target machine. This section is devoted to the study of the search space under this assumption. In section 3.1, structural completeness is introduced and the corresponding search space is introduced. An implicit representation of this search space in terms of state pair mergings is presented in section 3.2. Deterministic and unambiguous elements are respectively characterized in section 3.3 and 3.4.

3.1. Structural completeness

Definition 9. A sample $\mathcal{S}$ is said to be *structurally complete* with respect to a C-FSA M if there exists an acceptance of $\mathcal{S}$ such that:

- Every transition of M is exercised;
- Every state of M is used as an accepting state for each of its outputs i.e. $\forall q \in Q, \forall c \in \omega(q), \exists \sigma \in S_c, \hat{\delta}(q_s, \sigma) = q$.

$MCA(\mathcal{S})$ is the largest C-FSA with respect to which $\mathcal{S}$ is structurally complete. It is made of one branch per sequence and the C-tuple of accepted languages is exactly $\mathcal{S}$. $MCA(\mathcal{S})$ is generally a C-NFA, where the only possible non-deterministic transitions come from the start state q_s. Formally, we get:

Definition 10. The *maximal canonical C-FSA with respect to $\mathcal{S}$* is the C-FSA denoted by $MCA(\mathcal{S}) = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ constructed as follows:

- Σ is the alphabet on which $\mathcal{S}$ is defined;
- Γ is the alphabet of the C classes defined by $\mathcal{S}$;
- $Q = \{q_{c,i,j} | c \in \Gamma, i \in [1, |S_c|], j \in [1, |\sigma_{c,i}|]\} \cup \{q_s\}$;
- $\forall c \in \Gamma, \forall i \in [1, |S_c|],\ \omega(q_{c,i,|\sigma_{c,i}|}) = c$;
- $\forall a \in \Sigma,\ \delta(q_s, a) = \{q_{c,i,1} | a_{c,i,1} = a, c \in \Gamma, i \in [1, |S_c|]\}$;
- $\forall c \in \Gamma, \forall i \in [1, |S_c|], \forall j \in [1, |\sigma_{c,i}| - 1],\ \delta(q_{c,i,j}, a_{c,i,j+1}) = \{q_{c,i,j+1}\}$;

UA($\mathcal{S}$) is the C-FSA obtained by merging all the states of $MCA(\mathcal{S})$. For each non empty sample S_c, it accepts the language $L_c = \Sigma^*$, such that Σ is the alphabet on which is defined $\mathcal{S}$. For each empty sample $S_{c'}$, the accepted language is the empty language $L_{c'} = \emptyset$.

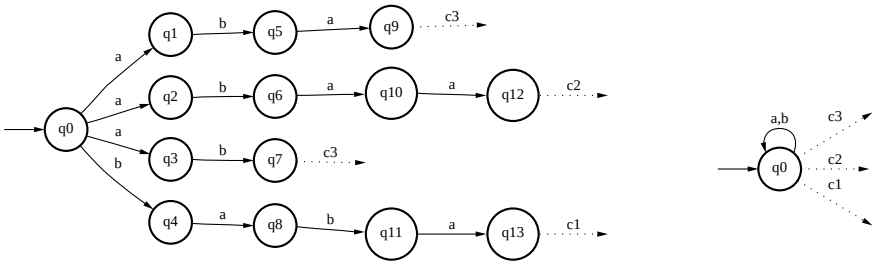

Figure 2. $MCA(\mathcal{S})$ and UA($\mathcal{S}$) when $\mathcal{S} = \langle S_{c1}, S_{c2}, S_{c3}\rangle = \langle\{baba\}, \{abaa\}, \{ab, aba\}\rangle$.

THEOREM 2 *Let $\mathcal{S}$ be a sample from a set of regular languages $\mathcal{L}$ and let M be a C-FSA accepting $\mathcal{L}$. If $\mathcal{S}$ is structurally complete with respect to M, then M belongs to $Lat(MCA(\mathcal{S}))$.*

Proof: This is an extension of the proof proposed in [3] based on [9].
Let $MCA(\mathcal{S}) = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ and let $M = (Q', q'_s, \Sigma, \Gamma, \delta', \omega')$.
A partition π will be defined such that M is isomorphic to $MCA(\mathcal{S})/\pi$.

Firstly, let us define $\Sigma_{c\in\Gamma}|S_c|$ sequences of states from the acceptance of $\mathcal{S}$ by M as follows: for each sequence $\sigma_{c,i}$ a sequence $(q'_{i,0}, \ldots, q'_{i,|\sigma_{c,i}|})$ of $|\sigma_{c,i}| + 1$ states is defined, where $q'_{i,0} = q'_s$ and $\forall j \in [0, |\sigma_{c,i}|], q'_{i,j+1} \in \delta'(q'_{i,j}, a_{c,i,j})$. Furthermore, $\forall c \in \Gamma, c \in \omega(q'i, |\sigma_{c,i}|)$. Next a function $\phi : Q \times Q'$ is defined as

1. $\phi(q_s) = q'_s$
2. $\forall i \in [1, \Sigma_{c\in\Gamma}|S_c|], \forall j \in [1, |\sigma_{c,i}|],\ \phi(q_{i,j}) = q'$, whenever $q' = q'_{i,j}$.

Let the partition π be given by $\forall q_k, q_l \in Q, B_\pi(q_k) = B_\pi(q_l) \Leftrightarrow \phi(q_k) = \phi(q_l)$. Then M is isomorphic to $MCA(\mathcal{S})/\pi$, since the structural completeness of $\mathcal{S}$ with respect to M implies the two following properties:

1. The next-state function of $MCA(\mathcal{S})/\pi$ exactly corresponds to δ' (since all transitions in δ' are exercised);
2. The output function of $MCA(\mathcal{S})/\pi$ exactly corresponds to ω' (since $\forall c \in \Gamma, \forall q' \in Q', c \in \omega'(q')\ \exists i \in [1, |S_c|]$ such that $q'_{i,|\sigma_{c,i}|}$).

Hence M belongs to $Lat(MCA(\mathcal{S}))$. ■

In the sequel of the article, we'll assume that the sample is structurally complete with respect to the target C-FSA. In this case, the hypothesis space considered is the lattice

of C-FSA whose null and universal elements are respectively $MCA(\mathcal{S})$ and $UA(\mathcal{S})$. Since each element M_i in the lattice may be represented as a C-FSA derived from $MCA(\mathcal{S})$ with respect to a partition π_i, an implicit representation of the hypothesis space may be composed of $MCA(\mathcal{S})$ and the set of partitions on its states (the partition lattice constructed for a four state $MCA(\mathcal{S})$ is depicted in figure 3). We call this implicit representation a *partition space of* $MCA(\mathcal{S})$.

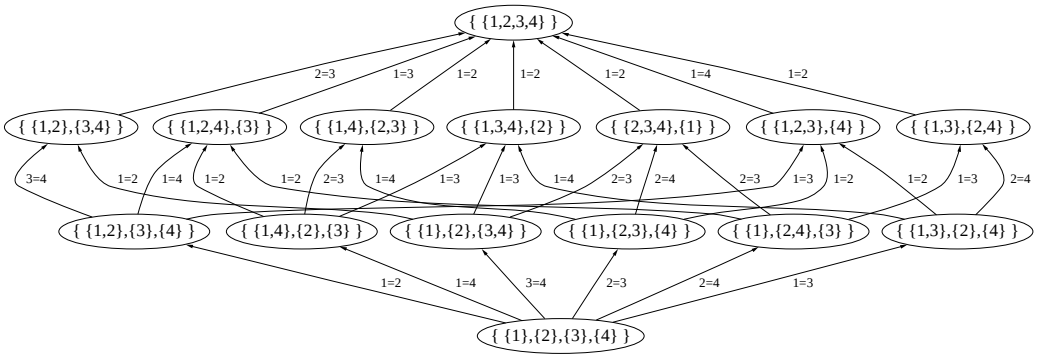

Figure 3. Partition space lattice for a 4 states $MCA(\mathcal{S})$.

3.2. State merging space

In the lattice, each partition may be reached by a succession of pair of states mergings. By changing the merging order or by considering different equivalent pairs in the blocks, several paths may exist. The particular study of the effects of merging two states motivates the introduction of a finer equational representation of the search space. We call *state merging space of* $MCA(\mathcal{S})$ the set of its state pairs considered as attributes with two possible values: $=$ if states should be merged in the derived C-FSA and $\neq$ if states should be in different blocks of the partition. This representation enables to work on incomplete assignments such that some values are not set for some pairs of states. This can be useful, for example, to specify implicitly a set of C-FSA in the lattice. On the contrary, if values has been assigned to all the pairs of states, the assignment is said to be *complete*. If in addition transitivity is satisfied, a complete assignment defines a partition and therefore represents an unique C-FSA in the lattice:

PROPOSITION 2 *A complete assignment on Q defines a partition if and only if:*

$$\forall q_1, q_2, q_3 \in Q,\ q_1 = q_2 \wedge q_2 = q_3 \Rightarrow q_1 = q_3$$

Proof: To define a partition, the relation $=$ has to be an equivalence relation. The $=$ relation is already reflexive and symmetrical. It has to be also transitive. This is what is stated in the proposition relation. ■

	q_1	q_2	q_3	q_4
q_1	$=$	$\neq$	$=$	?
q_2		$=$	$\neq$	$\neq$
q_3			$=$	?
q_4				$=$

(a)

	q_1	q_2	q_3	q_4
q_1	$=$	$\neq$	$=$	$\neq$
q_2		$=$	$\neq$	$\neq$
q_3			$=$	$=$
q_4				$=$

(b)

	q_1	q_2	q_3	q_4
q_1	$=$	$\neq$	$=$	$=$
q_2		$=$	$\neq$	$\neq$
q_3			$=$	$=$
q_4				$=$

(c)

Figure 4. Incomplete (a), complete (b), and transitive complete (c) assignment.

In the state merging space, we denote $B_=(q)$ the set of states q' such that $q = q'$ and $\delta_=$, $\omega_=$ the extensions, by the relation $=$, of respectively the next-state and the output functions defined by:

$$\delta_=(q,a) = \{B_=(q') \mid \exists q_e \in B_=(q), q' \in \delta(q_e,a)\}$$

$$\omega_=(q) = \bigcup_{q' \in B_=(q)} \omega(q')$$

PROPOSITION 3 *If the assignment defines a partition π , $\delta_=$ and $\omega_=$ define the next-state function δ' and the output function ω' of the C-FSA derived from $MCA(\mathcal{S}) = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ with respect to π:*

$$\forall B \in \pi, \forall q \in B, \forall a \in \Sigma,\ \delta'(B,a) = \delta_=(q,a),$$

$$\forall B \in \pi, \forall q \in B,\ \omega'(B) = \omega_=(q)$$

Proof: First, let us remark that if the assignment defines a partition π, transitivity of $=$ is satisfied, therefore $B_=(q)$ contains all the states of the block of q and thus equals $B_\pi(q)$. Then, the proposition results from the definition of a derived C-FSA and the definition of $\delta_=$ and $\omega_=$ which have been constructed to this end.

$$\begin{aligned} \forall B \in \pi, \forall a \in \Sigma,\ \delta'(B,a) &= \{B' \in \pi \mid \exists q \in B, \exists q' \in B', q' \in \delta(q,a)\} \\ &= \{B_=(q') \mid \exists q \in B, \exists q_e \in B_=(q), q' \in \delta(q_e,a)\} \\ &= \{\delta_=(q,a) \mid \exists q \in B\} \end{aligned}$$

Now the assignment defines a partition, therefore $\forall q_1, q_2 \in B, \delta_=(q_1,a) = \delta_=(q_2,a)$, so: $\forall B \in \pi, \forall q \in B, \forall a \in \Sigma,\ \delta'(B,a) = \delta_=(q,a)$.
Similarly:
$\forall B \in \pi, \omega'(B) = \bigcup_{q \in B} \omega(q) \Leftrightarrow \forall B \in \pi, \forall q \in B, \omega'(B) = \bigcup_{q_e \in B_=(q)} \omega(q_e) = \omega_=(q)$ ■

The extension to terminal state and last output functions is done accordingly:

$$\hat{\delta}_=(q,\epsilon) = B_=(q),\ \hat{\delta}_=(q,ma) = \{\delta_=(q',a) \mid q' \in \hat{\delta}_=(q,m)\}$$

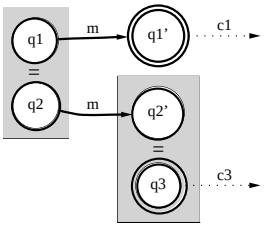

Figure 5. Example for $\delta_=$ and $\omega_=$.
In this configuration ($q_1 = q_2$ and $q'_2 = q_3$), $\delta_=(q_1, m) = \{q'_1, q'_2, q_3\}$ and $\omega_=(q'2) = \{c3\}$.

$$\hat{\omega}_=(q, m) = \{\omega_=(q') \mid q' \in \hat{\delta}_=(q, m)\}$$

According to proposition 3, a property is satisfied for a next-state or an output function of a C-FSA in $Lat(MCA(\mathcal{S}))$ only if this property is satisfied for the counterpart of the function in the merging state space. We focus now on two properties, determinism (section 3.3) and unambiguousness (section 3.4). For each of these properties, we proceed similarly. We first introduce an extension to C-FSA of results of regular inference on the search space presented by Dupont, Miclet and Vidal in [9]. Then, we propose a characterization of such properties in the state merging space in terms of constraints between pairs of states.

3.3. Determinism

To explore the $MCA(\mathcal{S})$ lattice, moving to a more general C-FSA is achieved by merging two states of the current C-FSA. To consider only deterministic C-FSA, the *deterministic merge operation* used in RPNI [19] may be used. This operation consists in merging the states giving rise to non determinism. It allows to obtain the lowest deterministic upper bound of the current C-FSA in the lattice which is eventually more general. This operation should not be confused with the classical determinization process [2] which produces an equivalent deterministic machine, but eventually outside the lattice.

Definition 11. Let $M = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ be a C-FSA. The *deterministic merge operation* may be stated as follows:

$$\text{while } (\exists q \in Q, \exists a \in \Sigma, \exists q_1, q_2 \in \delta(q, a), q_1 \neq q_2) \text{ merge(M, } q_1, q_2)$$

The result of this operation on $MCA(\mathcal{S})$ is a C-FSA called prefix tree acceptor.

Figure 6. Deterministic merge operation.

Definition 12. The *prefix tree acceptor* of $\mathcal{S}$, denoted $PTA(\mathcal{S})$, is the C-FSA derived from $MCA(\mathcal{S})/_\pi$ such that the partition π is defined to merge the states reached by the same prefix:

$$B_\pi(q) = B_\pi(q') \Leftrightarrow Pr(q) = Pr(q')$$

where $Pr(q)$ denotes the set of prefixes of a state q: $Pr(q) = \{u \in \Sigma^* | \hat{\delta}(q_s, u) = q\}$.

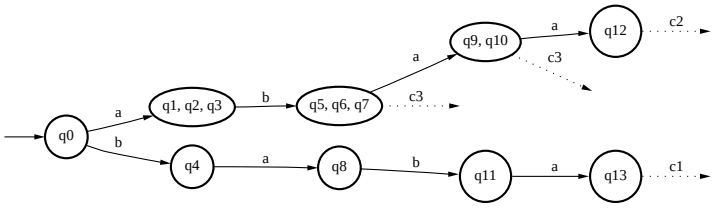

Figure 7. $PTA(\mathcal{S})$ (same training sample than in figure 2 for $MCA(\mathcal{S})$).

The prefix tree acceptor is by construction deterministic and accepts exactly $\mathcal{S}$. Since $PTA(\mathcal{S})$ belongs to $Lat(MCA(\mathcal{S}))$, we have the following property:

$$Lat(PTA(\mathcal{S})) \subseteq Lat(MCA(\mathcal{S})).$$

Another interesting property is that all the deterministic C-FSA in $Lat(MCA(\mathcal{S}))$ are included in $Lat(PTA(\mathcal{S}))$. Therefore, if the search is only concerned by deterministic C-FSA, $PTA(\mathcal{S})$ may be used as the null element of the lattice and the search may be restricted to this lattice. Moreover, we obtain the following theorem:

THEOREM 3 *Let $\mathcal{S}$ be a sample from a C-tuple of regular languages $\mathcal{L}$ and let $M(\mathcal{L})$ be the canonical C-DFA of $\mathcal{L}$. If $\mathcal{S}$ is structurally complete with respect to $M(\mathcal{L})$, then M belongs to $Lat(PTA(\mathcal{S}))$.*

Proof: The same argument than in the theorem 2 holds, except that, now, since M is deterministic, there is a unique acceptance of $\mathcal{S}$ from which a tree of states can be built that has the same structure than $PTA(\mathcal{S})$. ■

Though all C-DFA are included in $Lat(PTA(\mathcal{S}))$, some C-NFA are also included in this lattice. The deterministic merge operation can still be used to explore the space of C-DFA, but this operation can be avoided by choosing the two states to be merged:

PROPOSITION 4 *Let $\mathcal{S}$ be a sample from a set of regular languages $\mathcal{L}$ and let $M(\mathcal{L})$ be the canonical C-DFA of $\mathcal{L}$. If $\mathcal{S}$ is structurally complete with respect to $M(\mathcal{L})$, then a sequence of C-DFA exists such that: $PTA(\mathcal{S}) = PTA(\mathcal{S})/\pi_0 \preceq PTA(\mathcal{S})/\pi_1 \preceq \ldots \preceq PTA(\mathcal{S})/\pi_n = M(\mathcal{L})$ with $n \leq |Q_{PTA(\mathcal{S})}| - 1$.*

this proposition implies the following corollary:

COROLLARY 1 *Each C-DFA $M \in Lat(PTA(\mathcal{S}))$ may be reached by a succession of two states merging such that the corresponding sequence of derived C-FSA in $Lat(PTA(\mathcal{S}))$ is made of C-DFA.*

Proof: The proof of this proposition and its corollary for automata [8] is independent from acceptance or final state concepts, except the use of theorem 3. Thus, these results are also valid for C-DFA. ■

In the state merging space, the determinism may be handled by the following theorem related to the determinism of $\delta_=$ and therefore to the determinism of an eventually derived C-FSA.

THEOREM 4 *The function $\delta_=$ in the state merging space based on $MCA(\mathcal{S}) = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$ is deterministic if and only if:*

$$\forall q, q_1, q_2 \in Q, \forall a \in \Sigma, \forall q_1' \in \delta(q_1, a), \forall q_2' \in \delta(q_2, a) \ : \ q_1, q_2 \in B_=(q) \Rightarrow q_1' = q_2'$$

Proof: This relation is a translation in the state merging space of the deterministic merge operation. $\delta_=$ is deterministic: $\forall q \in Q, \forall a \in \Sigma, \mid \delta_=(q,a) \mid \leq 1$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \mid \{B_=(q') | \exists q_e \in B_=(q), q' \in \delta(q_e, a)\} \mid \leq 1$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \forall q_1, q_2 \in Q, q_1 = q, q_2 = q, \forall q_1' \in \delta(q_1, a), \forall q_2' \in \delta(q_2, a), q_1' = q_2'$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \forall q_1, q_2 \in B_=(q), \forall q_1' \in \delta(q_1, a), \forall q_2' \in \delta(q_2, a), q_1' = q_2'$ ■

If $MCA(\mathcal{S})$ is non deterministic, the application of this system yields a set of equality between pairs of states such that the corresponding set of potential C-FSA is $Lat(PTA(\mathcal{S}))$. To reduce the search space size and save computer resources, the state merging space may also be based, according to theorem 3, on $PTA(\mathcal{S})$ instead of $MCA(\mathcal{S})$. In this case the theorem 4, in which $MCA(\mathcal{S})$ is replaced by $PTA(\mathcal{S})$, still holds.

In this context, we may reconsider the two exploration strategies of the deterministic part of the search space. The first one, using deterministic merge operation, corresponds to a forward propagation of equality constraints of theorem 4. The second strategy, considering only two state mergings such that the resulting C-FSA is deterministic, amounts to avoid

such states merging implying forward propagation. It corresponds to a partial ordering of pairs of states such that a pair of state p_1 greater than a pair p_2 should not be merged before p_2. On the other hand, if the attribute of p_2 is assigned the value $\neq$, a backward propagation using modus-ponens may be investigated (figure 8).

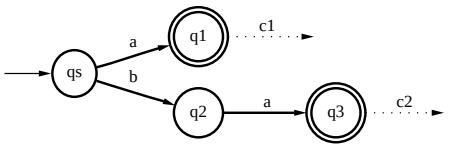

Figure 8. Partial ordering on pairs of states for cautious strategy.
For the considered PTA, determinism states that $q_s = q_2 \Rightarrow q_1 = q_3$. Therefore, to avoid intermediate C-NFA, q_1 and q_3 merging has to be considered before q_s and q_2 merging. If q_1 and q_3 are merged ($q_1 = q_3$) , the determinism relation is always satisfied. On the contrary, if these two states are separated ($q_1 \neq q_3$), then by propagation $q_s \neq q_2$, so q_s and q_2 should never be merged to ensure determinism.

3.4. Unambiguousness

PROPOSITION 5 *Let M and M' be two C-FSA such that $M \ll M'$*

- *If M' is unambiguous then M is unambiguous.*
- *if M is ambiguous then M' is ambiguous.*

Proof: These properties follow from language inclusion stated in prop 1. ■

As a consequence, unambiguous elements in a C-FSA lattice may be delimited by a *border set*, denoted by BS, [16] such that each C-FSA which can be derived in one element of BS is unambiguous. We introduce first the definition of an anti-string before giving the BS definition.

Definition 13. An *anti-string* $\overline{AS}$ in a lattice is a set of elements in the lattice such that any element of $\overline{AS}$ is not related by $\ll$ to any other element of $\overline{AS}$.

Definition 14. Let M be a C-FSA and $Lat(M)$ the associated lattice.
The *border set* BS_M is the anti-string composed of unambiguous C-FSA such that each C-FSA derived from one element of BS_M is ambiguous.

When the considered lattice is $Lat(MCA(\mathcal{S}))$, the border set $BS_{MCA(\mathcal{S})}$ sets the limit of generalization of unambiguous C-FSA. Thus, following the version space terminology [17], $BS_{MCA(\mathcal{S})}$ is the set G of *maximally general solutions*. In our case, the set S of *maximally*

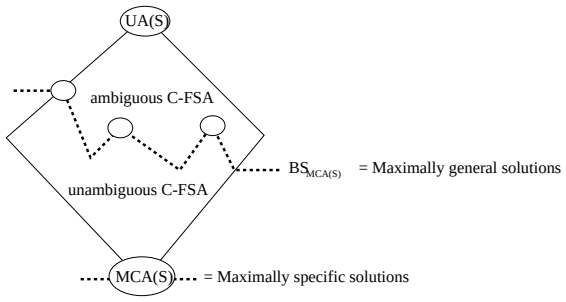

Figure 9. Unambiguous C-FSA search space.

specific solutions is reduced to $MCA(\mathcal{S})$. The set of unambiguous C-FSA with respect to which the sample is structurally complete is therefore characterized implicitly by $BS_{MCA(\mathcal{S})}$ (figure 9).

In the state merging space, the ambiguousness of the derived C-FSA is stated by the following theorem in which the $\omega_=(q'_1) \sim \omega_=(q'_2)$ notation has been introduced. This notation is equivalent to $\omega_=(q'_1) = \omega_=(q'_2) \vee \omega_=(q'_1) = \emptyset \vee \omega_=(q'_2) = \emptyset$, which means that the output of q'_1 is compatible with the output of q'_2.

THEOREM 5 *$\hat{\omega}_=$ is unambiguous if and only if:*

$$\begin{cases} \forall q \in Q, \exists c \in \Gamma,\ \omega_=(q) = \{c\} \vee \omega_=(q) = \emptyset \\ \forall q \in Q, \forall \sigma \in \Sigma^*, \forall q'_1, q'_2 \in \hat{\delta}_=(q,\sigma),\ \omega_=(q'_1) \sim \omega_=(q'_2) \end{cases}$$

Proof:

$\hat{\omega}_=$ is unambiguous $\Leftrightarrow \forall q \in Q, \forall \sigma \in \Sigma^*\ :\ |\hat{\omega}_=(q,\sigma)\ | \leq 1$

$\Leftrightarrow \forall q \in Q, \forall \sigma \in \Sigma^*, \exists c \in \Gamma,\ \hat{\omega}_=(q,\sigma) = \{c\}\ \vee\ \hat{\omega}_=(q,\sigma) = \emptyset$

$\Leftrightarrow \forall q \in Q, \forall \sigma \in \Sigma^*, \exists c \in \Gamma, \forall q' \in \hat{\delta}_=(q,\sigma),\ \omega_=(q') = \{c\}\ \vee\ \omega_=(q') = \emptyset$

Which can be split into two constraints: a unique classification for each reached states on the one hand, and, on the other hand, an identical classification for each state reached by the same sequence.

$$\begin{cases} \forall q \in Q, \forall \sigma \in \Sigma^*, \forall q' \in \hat{\delta}_=(q,\sigma), \exists c \in \Gamma,\ \omega_=(q') = \{c\} \vee \omega_=(q') = \emptyset \\ \forall q \in Q, \forall \sigma \in \Sigma^*, \forall q'_1, q'_2 \in \hat{\delta}_=(q,\sigma),\ \omega_=(q'_1) = \omega_=(q'_2) \vee \omega_=(q'_1) = \emptyset \vee \omega_=(q'_2) = \emptyset \end{cases}$$

By construction, each state of $MCA(\mathcal{S})$ or $PTA(\mathcal{S})$ can be reached, therefore the first relation has to be true for each state of Q; it can thus be rewritten as $\forall q \in Q, \exists c \in \Gamma, \omega_=(q) = \{c\} \vee \omega_=(q) = \emptyset$. ■

Let us notice that if the sample $\mathcal{S}$ is ambiguous, all the elements of the lattice based on $MCA(\mathcal{S})$ or $PTA(\mathcal{S})$ are ambiguous and the search of unambiguous C-FSA stops trivially. From here onwards, we assume for the search of unambiguous C-FSA that the sample is unambiguous. In this case $MCA(\mathcal{S})$ or $PTA(\mathcal{S})$ are unambiguous. Moreover, if we denote by Q the set of states of $MCA(\mathcal{S})$ or $PTA(\mathcal{S})$, then for each state q of Q, the classification is unique: $\exists c \in \Gamma, \omega(q) = \{c\} \vee \omega(q) = \emptyset$. Therefore the relation $\forall q_1, q_2 \in Q, q_1 = q_2, \omega(q_1) \sim \omega(q_2)$ implies the first relation in the system for unambiguousness. Since this relation is included in the second one where σ is the empty sequence, the system for unambiguousness may be reduced to the second relation. We have therefore the following corollary:

COROLLARY 2 *If the sample $\mathcal{S}$ is unambiguous, $\hat{\omega}_=$ is unambiguous if and only if:*

$$\forall q \in Q, \forall \sigma \in \Sigma^*, \forall q'_1, q'_2 \in \hat{\delta}_=(q, \sigma),\ \omega_=(q'_1) \sim \omega_=(q'_2)$$

This relation allows to define the *incompatible mergings set*, i.e. the set of pairs of states such that merging one of these pairs implies that the corresponding derived C-FSA are ambiguous:

$$\{\{q_1, q_2\} \mid q_1 = q_2 \Rightarrow \exists m \in \Sigma^*, \exists q'_1 \in \hat{\delta}_=(q_1, m), q'_2 \in \hat{\delta}_=(q_2, m),\ \omega_=(q'_1) \not\sim \omega_=(q'_2)\}$$

The computation of this set requires to merge each pair of states in order to know if the pair is incompatible. An approximation of the set of incompatible mergings may be obtained without merging the states beforehand. We denote $A_{\neq}$ the subset of incompatible mergings defined by:

$$A_{\neq} = \{\{q_1, q_2\} \mid \exists m \in \Sigma^*, \exists q'_1 \in \hat{\delta}_=(q_1, m), \exists q'_2 \in \hat{\delta}_=(q_2, m),\ \omega_=(q'_1) \not\sim \omega_=(q'_2)\}$$

$A_{\neq}$ is a good approximation in practice of the set of incompatible state merging. We propose here an algorithm (algorithm 1) to compute the initial $A_{\neq}$ set before any pair of state attribute has been assigned. The algorithm takes advantage of the property that $MCA(\mathcal{S})$ and $PTA(\mathcal{S})$ are tree structured and therefore, there exists only one in-going transition for each state q'. The functions returning respectively the symbol and the source state of in-going transition of q' are denoted by *In_Symbol(q')* and *Pred(q')*.

ALGORITHM 1 *Initialization of $A_{\neq}$*

1. **Initialization** *($A_{\neq}$, $M_0 = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$):*
2. *require $M_0 = MCA(\mathcal{S})$ or $PTA(\mathcal{S})$*
3. *$A_{\neq} \leftarrow \emptyset$*
4. *for all q_i in Q do*
5. *for all $q_j > q_i$ in Q do*
6. *if $\omega(q_i) \not\sim \omega(q_j)$ then*
7. **Separate_and_Propagate**$(\mathbf{q}_i, \mathbf{q}_j)$
8. *end if*

```
9.              end for
10.         end for
11.     end function

12.     Separate_and_Propagate(q_i,q_j):
13.         if {q_i ≠ q_j} ∉ A_≠ then
14.             /* Update A_≠ */
15.             A_≠ ← A_≠ ∪{ q_i ≠ q_j}
16.             /* Propagation */
17.             if In_Symbol(q_i) = In_Symbol( q_j) then
18.                 Separate_and_Propagate(Pred(q_i),Pred(q_j))
19.             end if
20.         end if
21.     end function
```

For each pair of states, the Separate_and_Propagate() procedure may be called only once, so the complexity of the $A_{\neq}$ initialization algorithm in terms of the number n of states of $MCA(\mathcal{S})$ or $PTA(\mathcal{S})$ is $O(n^2)$.

After initialization, values choices for some pairs of states may induce new incompatible state mergings. We sketch here how $A_{\neq}$ may be updated according to the theorem 2 after the choice of an equality value for a pair $\{q_1, q_2\}$. $q_1 = q_2$ implies that for each sequence σ in Σ^*, for each state q_1' and q_2' reached by σ from q_1 and q_2, a new compatibility constraints $\omega_=(q_1') \sim \omega_=(q_2')$ may be added. Furthermore, for each of these new compatibility constraint, if the output of state q_1' is nonempty when the output of q_2' is empty, then, for each state q_3 incompatible with q_1' (the output of q_3 is defined and different from q_1' output), q_2' and q_3 should not be merged together and this pair may be added to $A_{\neq}$.

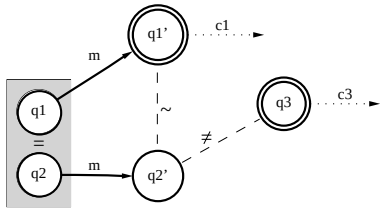

Figure 10. Consequence of choice $q_1 = q_2$ ($q_1 = q_2 \Rightarrow \omega_=(q_1') \sim \omega_=(q_2') \Rightarrow q_2' \neq q_3$)

4. Unambiguous C-DFA

4.1. Constraint system

The relations of the previous section theorems may be merged to define the set of unambiguous C-DFA. The corresponding system is composed of three relations and may be simplified because the deterministic merge procedure $q_1 = q_2 \Rightarrow q_1' = q_2'$ implies the equality of the output function $\omega_=(q_1') = \omega_=(q_2')$ when transitivity is assumed.

Thus, the set of unambiguous C-DFA may be implicitly characterized by a system of equality constraints between pairs of states expressed only in terms of the next-state and output functions of $PTA(\mathcal{S})$:

THEOREM 6 *Let $\mathcal{S}$ be a sample from a set of regular languages $\mathcal{L}$. The set of unambiguous C-DFA M, with respect to which $\mathcal{S}$ is structurally complete, is the set of C-FSA derived from $PTA(\mathcal{S})$ according to complete assignments satisfying the following system:*

$$\begin{cases} \forall q_1, q_2, q_3 \in Q \ : \ q_1 = q_2 \wedge q_2 = q_3 \Rightarrow q_1 = q_3 \\ \forall q_1, q_2 \in Q \ : \ q_1 = q_2 \Rightarrow \omega(q_1) \sim \omega(q_2) \\ \forall q_1, q_2, q_1', q_2' \in Q, \forall a \in \Sigma, q_1' \in \delta(q_1, a), q_2' \in \delta(q_2, a) \ : \ q_1 = q_2 \Rightarrow q_1' = q_2' \end{cases}$$

where $PTA(\mathcal{S}) = (Q, q_s, \Sigma, \Gamma, \delta, \omega)$

Proof: According to proposition 3, the assignments defining unambiguous C-DFA are those defining a partition such that $\hat{\delta}_=$ is deterministic and $\hat{\omega}_=$ is unambiguous. Since $\hat{\delta}_=$ is deterministic iff $\delta_=$ is deterministic, the relations to satisfy according to proposition 2, theorem 4 and corollary 2 are respectively:

$$\forall q_1, q_2, q_3 \in Q, \ q_1 = q_2 \wedge q_2 = q_3 \Rightarrow q_1 = q_3 \quad (1)$$
$$\forall q, q_1, q_2 \in Q, \forall a \in \Sigma, \forall q_1' \in \delta(q_1, a), \forall q_2' \in \delta(q_2, a), \ q_1, q_2 \in B_=(q) \Rightarrow q_1' = q_2' \quad (2)$$
$$\forall q \in Q, \forall \sigma \in \Sigma^*, \forall q_1', q_2' \in \hat{\delta}_=(q, \sigma), \ \omega_=(q_1') \sim \omega_=(q_2') \quad (3)$$

Assume that relation (1), i.e. transitivity is satisfied. In this case $\forall q \in Q, q_1, q_2 \in B_=(q)$ is equivalent to $\forall q_1, q_2 \in Q, q_1 = q_2$. Therefore, relation (2) may be simplified:

$$\forall q_1, q_2 \in Q, \forall a \in \Sigma, \forall q_1' \in \delta(q_1, a), \forall q_2' \in \delta(q_2, a), \ q_1 = q_2 \Rightarrow q_1' = q_2' \ (2a)$$

Now, if we assume that relation (1) and (2a) are satisfied, then $\delta_=$ is deterministic, therefore:

$$\forall q \in Q, \forall \sigma \in \Sigma^*, q_1', q_2' \in \hat{\delta}_=(q, \sigma) \Rightarrow q_1' = q_2'.$$

Conversely,

$$\forall q_1', q_2' \in Q, q_1 = q_2 \Rightarrow q_1', q_2' \in \hat{\delta}_=(q_1', \epsilon) \Rightarrow \exists q \in Q, \exists \sigma \in \Sigma^*, q_1', q_2' \in \hat{\delta}_=(q, \sigma).$$

The two sets of states are therefore identical, relation (3) may be simplified:

$$\forall q'_1, q'_2 \in Q, q'_1 = q'_2 \Rightarrow \omega_=(q'_1) \sim \omega_=(q'_2) \ (3a)$$

which is equivalent, under transitivity to:

$$\forall q_1, q_2 \in Q, q_1 = q_2 \Rightarrow \omega(q_1) \sim \omega_(q_2) \ (3b)$$

Relations (1), (2a) and (3b) compose the system of the theorem. ■

The system is stated as equality constraints between pairs of states which can be reversed by contraposition:

$$\begin{cases} \forall q_1, q_2, q_3 \in Q \ : \ q_1 \neq q_3 \Rightarrow q_1 \neq q_2 \vee q_2 \neq q_3 \\ \forall q_1, q_2 \in Q \ : \ \omega(q_1) \not\sim \omega(q_2) \Rightarrow q_1 \neq q_2 \\ \forall q_1, q_2, q'_1, q'_2 \in Q, \forall a \in \Sigma, q'_1 \in \delta(q_1, a), q'_2 \in \delta(q_2, a) \ : \ q'_1 \neq q'_2 \Rightarrow q_1 \neq q_2 \end{cases}$$

Before assigning a value to any pair of states, a set of inequalities for each pair of states whose output functions are incompatible comes from second relation. Then if the propagation by modus-ponens of inequalities according to the third relation is considered, a new set of inequalities have to be added . This set of inequalities corresponds to the initial set $A_{\neq}$ of incompatible mergings leading to ambiguous C-FSA defined at the end of section 3.4 and can be computed by algorithm 1.

4.2. Extended State Merging Algorithm

The constraint system in theorem 6 enables to characterize the set of unambiguous C-DFA in the lattice. It can be used as an implicit representation of this set, for example in an incremental process. But generally, the goal is to extract a set of unambiguous C-DFA with a given property or optimizing an evaluation function. We propose here a generic branch and bound algorithm (algorithm 2) to explore the search space and produce all the deterministic and unambiguous solutions of a minimization problem. Since each pair of states is assigned a value at each step and that "inequality" of states is taken into account, the ESMA algorithm ensure to visit only twice each element of the inference search space. The propagation of the constraints allows look-ahead and look-back techniques and, more generally, supply informations on the search space configuration.

The adaptation to a given problem may be done by acting on several tuning points. First, the completeness of the search ensured by the algorithm may be tuned. Since the problem may be NP-complete, the search may be limited to good solutions. The proposed algorithm allows to continue the search after each solution proposal, for example, if the user is not satisfied with it. Another way of limiting the search may be to stop the search when a given goal is reached. For example, the variable *best_min* may be initialized with a target value

ALGORITHM 2 *Extended State Merging Algorithm*

```
1.  ESMA(A):
2.      if Consistent(A) then
3.          if  Complete(A) then
4.              /* A is solution ? */
5.              if Eval(A) < best_min then
6.                  sol ← A
7.                  best_min ← Eval(A)
8.              else if Eval(A) = best_min then
9.                  sol ← sol ∪ A
10.             end if
11.         else if Partial_Eval(A) ≤ best_min then
12.             /* A is incomplete: set a new attribute value */
13.             (q1,q2) ← Choose_Pair_of_State(Q)
14.             A' ← A ∪ q1 = q2 ∪ Propagation(q1 = q2)
15.             ESMA(A')
16.             A' ← A ∪ q1 ≠ q2 ∪ Propagation(q1 ≠ q2)
17.             ESMA(A')
18.         end if
19.     end if
20. end function

21. main
22.     A ← Initialization(PTA(S))
23.     best_min ← min_upper_bound()
24.     ESMA(A)
25. end
```

and the search may be stopped at the first found solution.

A more difficult point is the compromise to find between the propagation of the constraints -done in the functions *Propagation*($q_1 = q_2$) and *Propagation*($q_1 \neq q_2$)- and the consistency checking -made by the function *Consistent*(*A*)-. This is the traditional dilemma between look-ahead and look-back strategies. Our experience aims at showing that maintaining transitivity and the set of incompatible mergings $A_{\neq}$ defined in section 3.4 is efficient for exact searches of set of solutions. Maintaining transitivity allows also to save space by pointing

out a representative of each block being constructed. It seems to be good compromise closer from the partition search space.

Another important point is the choice of the pairs of states to be considered with the *Choose_Pair_of_State(Q)* function. For heuristic search, the order in which the space is explored is obviously important, since the search space is not wholly explored. Heuristics related to the researched properties may be used, inductive biases proposed in [4] may also be considered to guide the inference. For exact searches, a good ordering enables to obtain quickly a good estimation of *best_min* and then allows more pruning. But the relation between the choice of the states and its propagation effects should also be carefully analyzed; good ordering should also decrease the search tree size. From this point of view, saturation techniques or entropy based techniques may be more adapted [7]. A mixed strategy may be employed: a first greedy strategy to obtain a good upper bound for *best_min* and then the second type of ordering to explore a smaller search tree.

The functions *Eval()*, which returns the solution evaluation to minimize, and *Partial_Eval()*, which returns a lower bound of this evaluation for an incomplete assignement, are both domain dependent. If the search aims at finding the minimal unambiguous C-DFA, the *Eval(A)* function returns the number of states induced by the assignement A. Concerning the *Partial_Eval()* function, maintaining a set of incompatible mergings, i.e. pair of states with an $\neq$ attribute value, is very useful. Then, a set of states that would never be merged together may be computed. This set of states is a *clique* (set of mutually adjacent states) such that adjacency between pair of states is true if the attribute value of the pair is $\neq$. A lower bound of the number of states for an incomplete assignement is thus given by the size of the biggest clique. This value can then be the value returned by the *Partial_Eval()* function. But, as this search has also been shown NP-complete, it may be more reasonable to return only an estimation of the biggest clique.

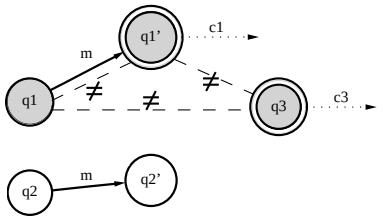

Figure 11. Clique of incompatible mergings.
As $q_1 \neq q'_1$, $q_1 \neq q_3$ and $q'_1 \neq q_3$, at least 3 states can not be merged with each other. Thus, for this assignment, the expected number of states of the derived C-FSA is greater or equal to 3.

5. Further research

We have introduced C-regular inference and studied the search space for the inference of deterministic and unambiguous classifiers. Considering both compatible and incompatible state merging allows to better understand the dynamic of the search space. The implicit representation of the search space and the characterization of the constraint propagation enables to build more efficient algorithms for an intensive exploration of the search space. The extension of this work to other finite state machine like transducers is an interesting challenge. For example, p-subsequential transducers [21], which are machines with states output but also transition output used in natural language processing [18] are indeed the closest finite state extension of C-FSA. They may be considered as belonging to the finite state classification framework (final output on accepting states) but the difficulty is that they also belong to the "interpretation" [13] framework (transition output).

Several algorithmic issues have also to be studied. Efficiency of different propagation strategies should be examined. Dependence between the choice of pair, its attribute value, the propagation of constraints and the pruning of the search is also an important field of investigation. This work may be done for the classical minimal machine search, but other optimization goals, such as classification criteria, may also be considered. For example, the search of machines separating, according to a distance between sequences, at most the given sample or the search of an unambiguous machine minimizing the number of ignored sequences could be studied, as well as the corresponding heuristics.

The implicit representation of the search space may be useful for other type of algorithms. Notably, meta-heuristic type of algorithms (Tabu search, Simulated Annealing, Genetic Algorithms) may take advantage from the simple definition of neighborhood and the possibility of building fitness functions according to the number of unsatisfied constraints. The versatility of the space may also be used with incomplete assignement representing set of C-FSA.

References

1. C. Higuera (de la), J. Oncina, and E. Vidal. Identification of dfa : data-dependent versus data-independant algorithms. *Grammatical inference Learning Syntax from Sentences, ICGI'96*, pages 311–325, 1996. Springer Verlag.
2. A. Aho and J. Ullman. *The Theory of Parsing, Translation and compiling, Vol 1 : Parsing.* Englewood Cliffs, Prentice-Hall, 1972.
3. R. Alquézar. *Symbolic and connectionist learning techniques for grammatical inference.* PhD thesis, Universitat Politecnica de Catalunya, March 1997.
4. R. Alquézar and A. Sanfeliu. Incremental grammatical inference from positive and negative data using unbiased finite state automata. In Shape, Structure and Pattern Recognition, *Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR'94, Nahariya (Israel)*, pages 291–300, 1995.
5. T. L. Booth. *Sequential Machines and Automata Theory.* John Wiley and Sons, New York, 3 edition, 1976.
6. F. Coste and J. Nicolas. Regular inference as a graph coloring problem. In *Workshop on Grammar Inference, Automata Induction, and Language Acquisition (ICML' 97), Nashville, TN.*, 1997.

7. F. Coste and J. Nicolas. How considering incompatible state mergings may reduce the dfa induction search tree. In Vasant Honavar and Giora Slutzki, editors, *Fourth International Colloquium on Grammatical Inference (ICGI-98), Ames, Iowa, USA*, volume 1433 of *LNAI*, pages 199–210, Berlin, July 1998. Springer Verlag.
8. P. Dupont. *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue.* PhD thesis, Ecole Nationale Supérieure des Télécommunications, 1996.
9. P. Dupont, L. Miclet, and E.Vidal. What is the search space of the regular inference ? *ICGI'94, Grammatical inference and Applications*, pages 25–37, 1994. Springer Verlag.
10. K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey — part I and II. *IEEE-Transactions on Systems, Man and Cybernetics*, 5:95–111 and 409–423, 1975.
11. E. M. Gold. Langage indentification in the limit. *Information and control*, 10(5):447 – 474, 1967.
12. J. Gregor. Data-driven inductive inference of finite-state automata. *Intenationnal Journal of Pattern Recognition and Artificial Intelligence*, 8(1):305 – 322, 1994.
13. E. Vidal J. Oncina, P. García. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15:448 – 458, 1993.
14. K. J. Lang. Random dfa's can be approximately learned from sparse uniform examples. *5th ACM workshop on Computation Learning Theorie*, pages 45 – 52, 1992.
15. K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, 1433:1–12, 1998.
16. L. Miclet and C. de Gentille. Inférence grammaticale à partir d'exemples et de contre-exemples : deux algorithmes optimaux : (big et rig) et une version heuristique (brig). *JAVA94, Journées Acquisition, Validation, Apprentissage*, pages F1–F13, 1994. Strasbourg France.
17. T. Mitchell. *Version Spaces : an approach to concept learning.* PhD thesis, Standford University, dec 1978.
18. Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(4), 1997.
19. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, pages 49 – 61, 1992.
20. R. Parekh and V. Honavar. An incremental interactive algorithm for regular grammar inference. In L. Miclet and C. de la Higueira, editors, *ICGI96, Grammatical Inference: Learning Syntax from Sentences*, volume 1147 of *Lecture Notes in Artificial Intelligence*, pages 238–249. Springer Verlag, September 1996.
21. M. P. Schützenberger. Sur une variante des fonctions sequentielles. *Theoretical Computer Science*, 4(1):47–57, February 1977.
22. B. Trakhenbrot and Ya. Barzdin. Finite automata : Behavior and synthesis. *Amsterdam, North Holland Pub. Comp*, 1973.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399